A Study & Learn Guide: Installing DifferentialEquations.jl and ModelingToolkit.jl Without Hassle

Prepared for learners at Beginner, Intermediate, and Advanced Levels

Introduction

DifferentialEquations.jl and ModelingToolkit.jl form the backbone of scientific computing in Julia. They power digital twins, simulations, hydropower models, control systems, and PDE/DAE modeling.

Yet many learners struggle with installation issues: broken dependencies, corrupted artifacts, incompatible BLAS libraries, or global environments that become polluted over time.

This study guide explains, step by step, how to install the *lightest* and most stable versions of these packages, suitable for any computer.

Throughout the document you will encounter rhetorical questions that mimic the process of self-inquiry: *What environment am I working in? Why do global installs break?* These checkpoints reinforce learning.

This guide serves **all levels**:

- **Beginners** learn clean installation and confidence using Julia.
- **Intermediate** users understand why environments break.
- **Advanced** users learn how to create reproducible scientific stacks.

1 Why Installation Fails

Before learning the correct method, consider: *Where do installation problems usually come from?*

Common causes include:

- Installing packages in the global environment `@v1.X`
- Mixing MKL and OpenBLAS unintentionally
- Using older versions of Julia (pre-1.10)
- Corrupted artifact downloads on Windows
- Conflicting dependencies from unrelated projects
- Antivirus tools blocking downloads silently

This raises a natural question: *Can these problems be avoided entirely?* Yes — by using project-specific environments.

2 The Golden Rule: Never Install in the Global Environment

Julia isolates code using *project environments*. This makes scientific work reproducible.

Ask yourself: *If my global environment contains hundreds of packages, should I install complex tools like ModelingToolkit into it?*

The answer is always **no**. Instead, create a clean project:

```
mkdir mtk_project
cd mtk_project
julia --project=.
```

Inside Julia:

```
using Pkg
Pkg.activate(".")
Pkg.add(["DifferentialEquations", "ModelingToolkit"])
```

You now have an isolated, clean environment. Nothing else on your system can interfere with it.

Checkpoint Question

Why does activating a project prevent global dependency conflicts?

Reflecting on this helps learners internalize Julia's package philosophy.

3 The Light Version: Minimal MTK Stack

Beginners and lightweight users often do not need PDE solvers, stochastic solvers, GPU offloading, or large symbolic dependencies.

Ask yourself: *Do I truly need the entire ModelingToolkit ecosystem, or only ODE/DAE tools?*

A minimal installation looks like this:

```
Pkg.add([
    "ModelingToolkit",
    "SciMLBase",
    "OrdinaryDiffEq",
    "NonlinearSolve",
    "Symbolics"
])
```

This version:

- Installs quickly on all operating systems
- Avoids heavy dependencies
- Minimizes artifact downloads
- Reduces the risk of installation failure

Checkpoint Question

Which dependency categories can I remove safely for my project?
This develops good software engineering intuition.

4 Cleaning, Repairing, and Rebuilding an Environment

Even experts occasionally encounter corrupted artifacts. How do we repair an installation safely?

```
using Pkg
Pkg.update()
Pkg.precompile()
Pkg.instantiate()
Pkg.gc()
```

If a specific package breaks:

```
Pkg.rm("DifferentialEquations")
Pkg.add("DifferentialEquations")
```

If artifacts are corrupted:

```
Pkg.build()
```

Note: This process is fully reproducible and does not depend on the computer's hardware.

5 Should You Upgrade Your Computer?

Learners often ask: *Is my computer too weak to run ModelingToolkit? Should I upgrade?*
In most cases, the answer is **no**. MTK and DifferentialEquations run on almost any modern laptop.

Minimum Requirements

- 8 GB RAM
- Any CPU since 2015
- macOS, Linux, or Windows 10/11

Recommended for Large Models

- 16–32 GB RAM
- SSD storage
- 8+ CPU threads
- Linux or WSL2 (Ubuntu) for maximum stability

Checkpoint Question

What class of models do I want to build? Symbolic PDEs or simple ODEs?
This helps learners match hardware to needs.

6 A Minimal Working Example

Once installation succeeds, confirm that everything works:

```
using ModelingToolkit, OrdinaryDiffEq

@variables x(t)
@parameters a = 1.0
D = Differential(t)

eq = D(x) ~ a * x
sys = ODESystem(eq)

prob = ODEProblem(sys, [x => 1.0], (0.0, 5.0))
sol = solve(prob, Tsit5())

sol(5)
```

This test validates:

- symbolic construction,
- system transformation,
- numeric solver availability,
- successful installation.

7 A Stable Setup for All Learners

The following stack is consistently reliable:

- Julia 1.10 or 1.11
- VS Code + Julia extension
- Project-specific environments
- OpenBLAS or MKL (not both)
- Optional: WSL2 Ubuntu for Windows users

Reflect once more: *Is my workflow reproducible and isolated, or shared and fragile?*

Conclusion

Installing scientific packages should not be stressful. By using project environments, minimal stacks, and stable workflows, any learner—from beginner to advanced—can work with DifferentialEquations.jl and ModelingToolkit.jl without hassle.

This document encourages active learning through self-questioning, reinforcing deeper understanding rather than passive reading.

Acknowledgements

This learning document was created using an AI-assisted workflow. The concepts, explanations, and structure were generated using **ChatGPT 5.1**, guided by iterative prompts exploring installation stability, reproducible environments, and efficient scientific computing practices. The content was refined through interactive question-driven learning, aligned with the Study & Learn mode of the platform.